# DATA STRUCTURES

# LIST

# INTRODUCTION

- List is a sequence of values called items or elements.
- The elements can be of any data type.
- The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets.
- List are mutable, meaning, their elements can be changed.

# CREATING A LIST?

- In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).

**Method -1 without constructor**

```
# empty list
my_list = []
# list of integers
my_list = [1, 2, 3]
# list with mixed datatypes
my_list = [1, "Hello", 3.4]
# nested list
my_list = [“welcome", [8, 4, 6]]
```

**Method-2 using list constructor**

```
# empty list
my_list = list()
# list of integers
my_list = list([1, 2, 3])
```

# ACCESS ITEMS FROM A LIST

- It can be access in several ways
- Use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.
- Example:

```
list = ['p','r','o','b','e']

print(list[2]) #Positive Indexing

print(list[-2]) #Negative Indexing
```

Output

```
O
b
```

# ACCESSING THE ELEMENTS OF A LIST

- Index operator [] is used to access an item in a list. Index starts from 0
- marks=[90,80,50,70,60]
- print(marks[0])

  Output: 90
- Nested list:
- my_list = [“welcome", [8, 4, 6]]
- Print(marks[1][0])

  Output: 8

# ACCESSING - NEGATIVE INDEXING

- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on

```
my_list = ['p','r','o','b','e']

# Output: e
print(my_list[-1])

# Output: p
print(my_list[-5])
```

# HOW TO CHANGE OR ADD ELEMENTS TO A LIST?

#Change Elements

= operator with index

```
>>> marks=[90,60,80]
>>> print(marks)
[90, 60, 80]
>>> marks[1]=100
>>> print(marks)
[90, 100, 80]
```

#Add Elements

- add one item to a list using append() method
- add several items using extend()
- insert one item at a desired location by using the method insert()

```
>>> marks.append(50)
>>> print(marks)
[90, 100, 80, 50]

>>> marks.extend([60,80,70])
>>> print(marks)
[90, 100, 80, 50, 60, 80, 70]

>>> marks.insert(3,40)
>>> print(marks)
[90, 100, 80, 40, 50, 60, 80, 70]
```

# HOW TO DELETE OR REMOVE ELEMENTS FROM A LIST?

- delete one or more items from a list using the keyword del.
- It can even delete the list entirely.

```
>>> print(marks)
[90, 100, 80, 40, 50, 60, 80, 70]

>>> del marks[6]
>>> print(marks)
[90, 100, 80, 40, 50, 60, 70]
>>> del marks
>>> print(marks)
Name Error: name 'marks' is not defined
```

- clear() method to empty a list.

```
>>> marks.clear()
>>> print(marks)
[]
```

- remove() method to remove the given item

```
>>> marks=[90,60,80]
>>> marks.remove(80)
>>> print(marks)
[90, 60]
>>> marks.remove(100)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    marks.remove(100)
ValueError: list.remove(x): x not in list
```

- pop() method to remove an item at the given index.

```
>>> marks=[100,20,30]
>>> marks.pop()
30
>>> print(marks)
[100, 20]
>>> >>> marks.pop(0)
100
>>> print(marks)
[20]
```

delete items in a list by assigning an empty list to a slice o elements.

```
marks=[100,20,30]
>>> marks[1:2]=[]
>>> print(marks)
[100, 30]
```

- >>> marks.extend([40,50,60,70])
- >>> marks
- [90, 40, 50, 60, 70]
- >>> marks.pop()
- 70
- >>> marks.pop()
- 60

# LIST OPERATORS

- Slicing [::] (i.e)  list[start:stop:step]
- Concatenation = +
- Repetition= *
- Membership = in
- Identity = is

## SLICE LISTS

- Accessing a range of items in a list by using the slicing operator [ ] using (colon :).
- Slicing can be best visualized by considering the index to be between the elements.
- ✓ Example:

```
list = ['p','r','o','b','e']

print(list[0:4]) #Positive

print(list[-2:-1]) # Negative
```

**Output**

```
['p', 'r', 'o', 'b']
['b']
```

# PYTHON LIST METHODS

| |
|---|
| **append()** - Add an element to the end of the list |
| **extend()** - Add all elements of a list to the another list |
| **insert()** - Insert an item at the defined index |
| **remove()** - Removes an item from the list |
| **pop()** - Removes and returns an element at the given index |
| **clear()** - Removes all items from the list |
| **index()** - Returns the index of the first matched item |
| **count()** - Returns the count of number of items passed as an argument |
| **sort()** - Sort items in a list in ascending order |
| **reverse()** - Reverse the order of items in the list |
| **copy()** - Returns a shallow copy of the list |

# LIST METHODS

- append() - Add an element to the end of the list.
- count() - Returns the count of number of items passed as an argument.
- extend() - Add all elements of a list to the another list.
- index() - Returns the index of the first matched item.
- insert() - Insert an item at the defined index.
- pop() - Removes and returns an element at the given index.
- copy() - Returns a shallow copy of the list
- remove() - Removes an item from the list.
- reverse() - Reverse the order of items in the list.
- sort() - Sort items in a list in ascending order.

# APPEND() METHODS

- The append() method adds a single item to the existing list.
- The append() method only modifies the original list. It doesn't return any value.
- The append() method takes a single *item* and adds it to the end of the list.
- The *item* can be numbers, strings, another list, dictionary etc.
- Syntax

```
list.append(item)
```

# APPEND() METHODS - PROGRAMS

**Example 1: Adding Element to a List**

```
list = ['hi', 'hello']

print('old list: ', list)

list.append('welcome!')

print('Updated list: ', list)
```

**Output**

```
old list: ['hi', 'hello']
Updated list: ['hi', 'hello','welcome!']
```

**Example 2: Adding List to a List**

```
list = ['hi', 'hello']

print('old list: ', list)

list1=['welcome']

list.append(list1)

print('Updated list: ', list)
```

**Output**

```
old list: ['hi', 'hello']
Updated list: ['hi', 'hello', ['welcome']]
```

# COUNT() METHODS

- Count() method counts how many times an element has occurred in a list and returns it.
- The count() method takes a single argument:
  - **element** - element whose count is to be found.
- The count() method returns the number of occurrences of an element in a list.
- Syntax

```
list.count(element)
```

# COUNT() METHODS - PROGRAMS

**Example: Count the occurrence of an element in the list**

```
list = ['h','i','h','o','w','y','o','u','!','!']

print('The count of i is:',list.count('i'))

print('The count of o is:',list.count('o'))
```

Output

```
The count of i is: 1
The count of o is: 2
```

**Example 2: Count the occurrence of tuple and list inside the list**

```
list = ['a', ('h', 'i'), ('a', 'b'), [8, 4]]

count = list.count(('a', 'b'))

print("The count of ('a', 'b') is:", count)

count = list.count([3, 4])

print("The count of [3, 4] is:", count)
```

Output

```
The count of ('a', 'b') is: 1
The count of [3, 4] is: 0
```

# EXTEND() METHODS

- The extend() extends the list by adding all items of a list to the end.
- Extend() method takes a single argument (a list) and adds it to the end.
- This method also add elements of other native data_types ,like tuple and set to the list,
- Syntax

```
list1.extend(list2)
```

Note : The elements of *list2* are added to the end of *list1*.

```
list.extend(list(tuple_type))
```

Note : The elements of *tuple* are added to the end of *list1*.

# EXTEND() METHODS - PROGRAMS

**Example 1: Using extend() Method**

```
language = ['C', 'C++', 'Python']

language1 = ['JAVA', 'COBOL']

language.extend(language1)

print('Language List: ', language)
```

Output

```
Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL']
```

# EXTEND() METHODS - PROGRAMS

**Example 2: Add Elements of Tuple and Set to List**

```
language = ['C', 'C++', 'Python']
language_tuple = ['JAVA', 'COBOL']
language_set = {'.Net', 'C##'}
language.extend(language_tuple)
print('New Language List: ', language)
language.extend(language_set)
print('Newest Language List: ', language)
```

Output

```
New Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL'] Newest
Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL',
'.Net', 'C##']
```

# INDEX( ) METHODS

- Index() method search and find given element in the list and returns its position.
- However, if the same element is present more than once, index() method returns its smallest/first position.
- Index in Python starts from 0 not 1.
- The index() method returns the index of the element in the list.
- The index method takes a single argument:
  - **element** - element that is to be searched.
- Syntax

```
list.index(element)
```

# INDEX( ) METHODS - PROGRAMS

**Example 1: Find position of element present in the list and not present in the list**

```
list = ['h','i','h','o','w','y','o','u','!','!']

print('The count of i is:',list.index('i'))

print('The count of o is:',list.index('o'))

print('The count of o is:',list.index('z'))
```

Output

```
The count of i is: 1

The count of o is: 3

ValueError: 'z' is not in list
```

# INSERT( ) METHODS

- The insert() method inserts the element to the list at the given index.
- The insert() function takes two parameters:
  - **index** - position where element needs to be inserted
  - **element** - this is the element to be inserted in the list
- The insert() method only inserts the element to the list. It doesn't return any value.
- Syntax

```
list.insert(index, element)
```

# INSERT( ) METHODS - PROGRAMS

**Example 1: Inserting Element to List**

```
vowels=['a','i','o','u']

print("old list =",vowels)

vowels.insert(1,'e')

print("new list =",vowels)
```

Output

```
old list = ['a', 'i', 'o', 'u']
new list = ['a', 'e', 'i', 'o', 'u']
```

# INSERT( ) METHODS - PROGRAMS

**Example 2: Inserting a Tuple (as an Element) to the List**

```
list = [{1, 2}, [5, 6, 7]]

print('old List: ', list)

number_tuple = (3, 4)

list.insert(1, number_tuple)

print('Updated List: ', list)
```

Output

```
old List: [{1, 2}, [5, 6, 7]]
Updated List: [{1, 2}, (3, 4), [5, 6, 7]]
```

# POP( ) METHODS

- The pop() method takes a single argument (index) and removes the element present at that index from the list.
- If the index passed to the pop() method is not in the range, it throws **IndexError: pop index out of range** exception.
- The parameter passed to the pop() method is optional. If no parameter is passed, the default index **-1** is passed as an argument.
- Also, the pop() method removes and returns the element at the given index and updates the list.
- Syntax

```
list.pop(index)
```

# POP( ) METHODS - PROGRAMS

**Example 1:** Pop Element at the Given Index from the List

```
list = ['Python', 'Java', 'C++', 'French', 'C‘]

return_value = language.pop(3)

print('Return Value: ', return_value)

print('Updated List: ', language)
```

Output

```
old list = ['Python', 'Java', 'C++', 'French', 'C']
Return Value: French
Updated List: ['Python', 'Java', 'C++', 'C']
```

# POP( ) METHODS - PROGRAMS

**Example 2: pop() when index is not passed**

```
language = ['Python', 'Java', 'C++', 'C']
print('Return Value: ', language.pop())
print('Updated List: ', language)
print('Return Value: ', language.pop(-1))
print('Updated List: ', language)
```

Output

```
Return Value: C
Updated List: ['Python', 'Java', 'C++']
Return Value: C++ Updated List:
['Python', 'Java']
```

# COPY( ) / ALIASING METHODS

- The copy() method returns a shallow copy of the list.
- A list can be copied with **=** operator.

```
old_list = [1, 2, 3]

new_list = old_list
```

- The problem with copying the list in this way is that if you modify the *new_list*, the *old_list* is also modified.
- Syntax

```
new_list = old_list
```

# COPY( ) / ALIASING METHODS - PROGRAMS

**Example 1: Copying a List**

```
old_list = [1, 2, 3]

new_list = old_list

new_list.append('a')

print('New List:', new_list )

print('Old List:', old_list )
```

Output

```
New List: [1, 2, 3, 'a']
Old List: [1, 2, 3, 'a']
```

# SHALLOW COPY( ) / CLONING METHODS

- We need the original list unchanged when the new list is modified, you can use copy() method.
- This is called **shallow copy**.
- The copy() method doesn't take any parameters.
- The copy() function returns a list. It doesn't modify the original list.
- Syntax

```
new_list = list.copy()
```

# SHALLOW COPY( ) / CLONING METHODS - PROGRAMS

## Example 2: Shallow Copy of a List Using Slicing

```
list = [1,2,4]
new_list = list[:]
new_list.append(5)
print('Old List: ', list)
print('New List: ', new_list)
```

**Output**

Old List: [1, 2, 4]

New List: [1, 2, 4, 5]

# REMOVE( ) METHODS

- The remove() method searches for the given element in the list and removes the first matching element.
- The remove() method takes a single element as an argument and removes it from the list.
- If the element(argument) passed to the remove() method doesn't exist, valueError exception is thrown.
- Syntax

```
list.remove(element)
```

# REMOVE( ) METHODS - PROGRAMS

**Example 1 :Remove Element From The List**

```
list = [5,78,12,26,50]

print('Original list: ', list)

list.remove(12)

print('Updated list elements: ', list)
```

Output

Original list: [5, 78, 12, 26,

50]Updated list: [5, 78, 26, 50]

# REVERSE( ) METHODS

- The reverse() method reverses the elements of a given list.
- The reverse() function doesn't return any value. It only reverses the elements and updates the list.
- Syntax

```
list.reverse()
```

# REVERSE( ) METHODS - PROGRAM

## Example 1 :Reverse a List Using Slicing Operator

```
list = [1,5,8,6,11,55]

print('Original List:', list)

reversed_list = list[::-1]

print('Reversed List:', reversed_list)
```

Output

Original List: [1, 5, 8, 6, 11, 55]

Reversed List: [55, 11, 6, 8, 5, 1]

## Example 2: Reverse a List

```
list = [1,5,8,6,11,55]
print('Original List:', list)
list.reverse()
print('Reversed List:', list)
```

Output

Original List: [1, 5, 8, 6, 11, 55]

Reversed List: [55, 11, 6, 8, 5, 1]

# SORT( ) METHODS

- The sort() method sorts the elements of a given list.
- The sort() method sorts the elements of a given list in a specific order - Ascending or Descending.
- Syntax

```
list.sort()
```

# REVERSE( ) METHODS - PROGRAM

**Example 1 :Sort a given List in ascending order**

```
list = [1,15,88,6,51,55]

print('Original List:', list)

list.sort()

print('sorted List:', list)
```

Output

Original List: [1, 15, 88, 6, 51, 55]

sorted List: [1, 6, 15, 51, 55, 88]

**Example 2 : Sort the list in Descending order**

```
list = [1,15,88,6,51,55]

print('Original List:', list)

list.sort(reverse=True)

print('sorted List:', list)
```

Output

Original List: [1, 15, 88, 6, 51, 55]

sorted List: [88, 55, 51, 15, 6, 1]

# BUILT-IN FUNCTIONS WITH LIST

| Function | Description |
|---|---|
| all() | Return True if all elements of the list are true (or if the list is empty). |
| any() | Return True if any element of the list is true. If the list is empty, return False. |
| enumerate() | Return an enumerate object. It contains the index and value of all the items of list as a tuple. |
| len() | Return the length (the number of items) in the list. |
| list() | Convert an iterable (tuple, string, set, dictionary) to a list. |
| max() | Return the largest item in the list. |
| min() | Return the smallest item in the list |
| sorted() | Return a new sorted list (does not sort the list itself). |
| sum() | Return the sum of all elements in the list. |
| copy() | |
| reversed() | |